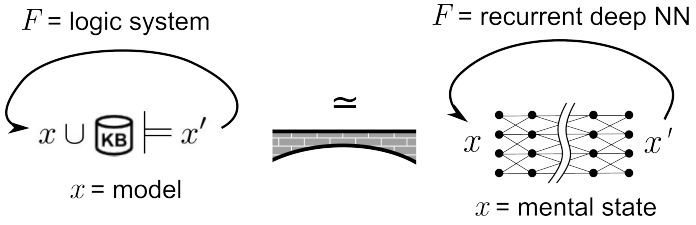
A bridge between logic and neural

甄景贤 (King-Yin Yan)

General.Intelligence@Gmail.com

Abstract. Logic-based inductive learning is based on slow combinatorial search; the bottleneck of classical AI. Neural network learning is based on gradient descent, which is much faster. We examine the abstract structure of logic (the consequence operator $\vdash$) and try to transfer this structure to the neural-network setting.

This problem took a long time to solve because the structure of logic is not easily expressible as a mathematical structure. The perspective of **model theory** is helpful in understanding this connection:



(1)

I will first explain the structure of the neural side, then the structure of the logic side.

1 The structure of neural networks

A **neural network** is basically:

weight matrix for each layer total # of layers

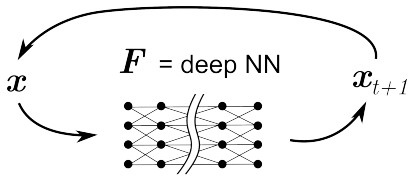
$$F(\boldsymbol{x}) = \sigma(W_1\sigma(W_2...\sigma(W_L\,\boldsymbol{x}))) \tag{2}$$

where σ is the sigmoid function applied component-wise to each neuron (whose role is to provide **non-linearity**).

σ acts on each component of $\boldsymbol{x}$; Its action is **not invariant** under a change of coordinates. Therefore, σ is not a vector operation, and thus $\mathbb{X}$ is not a **vector space** structure. Common practice is to denote $\vec{x}$ as a vector, but this is misleading.

Consider, for example, to recognize *"white cat chasing black cat"*. The *"cat"* object has to be recognied twice; Obviously we should not waste 2 neural networks to do this. If we connect a neural network **head to tail**, we get a minimalist cognitive architecture with a **recurrent**

loop:



(3)

Its state equation is:

$$\boxed{\text{discrete time}} \qquad \boldsymbol{x}_{t+1} = \boldsymbol{F}(\boldsymbol{x}_t) \tag{4}$$

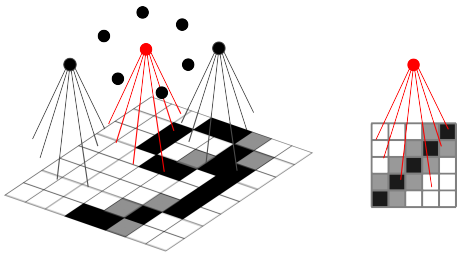
$$\boxed{\text{continuous time}} \qquad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{5}$$

From this we can see that $\mathbb{X}$ is a **differentiable manifold**. The deeper theory is that it is a Hamiltonian system, having a **symplectic** structure (cf. our first paper [12]).

1.1 What are "features"?

Now let's think about how a neural network performs pattern recognition, perhaps it would be illuminating....

Consider the simplest case, eg. a layer of neural network extracting visual features from the digit "9". This layer may have many neurons (left figure), each neuron locally covers a region of the input layer, the so-called "local receptive field" in the neuroscience of vision (right figure).



(6)

Suppose the red neuron is responsible for recognizing the "diagonal line" feature. Its equation is $\boldsymbol{y} = \sigma(W\boldsymbol{x})$. The matrix W's role is to affine "rotate" the feature space, so that the features we desire is pointing in a certain direction. Then we use σ to "**squeeze**" the desired and undesired features. The output after σ represents the **presence or absence** of a particular feature, ie $\{0, 1\}$. This is a form of information **compression**.

From the above reasoning, we may draw a general principle, which I call the **Neural Postulate**:

- Each neuron represents the presence or absence of a certain **feature**
- Higher-layer neurons represents **relations** between lower-layer features

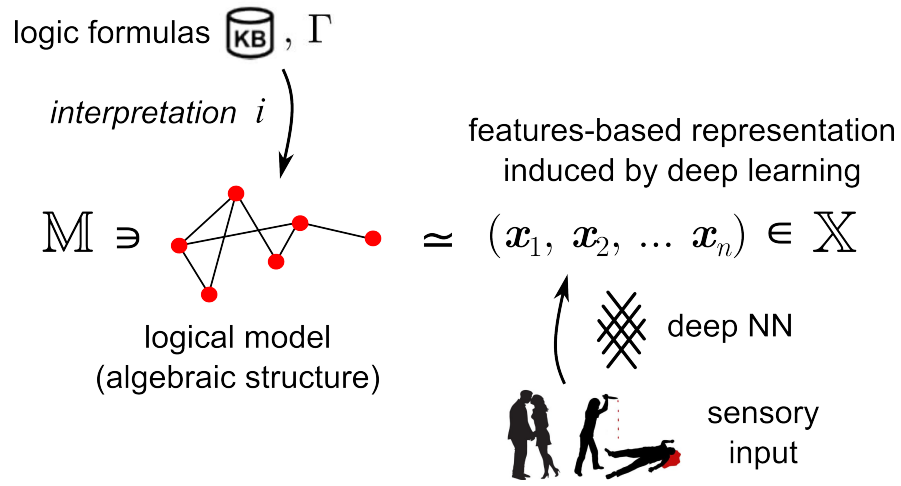
(7)

Following this line of thinking, we may conjecture this correspondence:

$\mathcal{M}$	$\simeq$	$\mathcal{X}$
constant	$\Leftrightarrow$	neuron
relation 	$\Leftrightarrow$	relation between higher and lower neurons

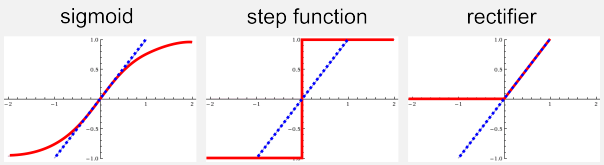
(8)

The following cartoon illustrates this correspondence from the perspective of **model theory**:



(9)

A little digression on chaos theory:
The role of $\int^{-1}$ is to "stretch", dragging points that are close neighbors to more distant positions. Looking at the common **threshold functions**, we see they are all non-linear **deformations** away from the identity $y = x$:



(10)

This is very similar to the "horseshoe" proposed by Steven Smale [9], a recipe for creating chaos. A variant of the Smale horseshoe is the "baker map", analogous to kneading dough in bakery. "Stretching" and then putting back to the original space, and repeating this process, creates chaos [2] [10].

2 Structure of logic

A **logic system** can be defined as:

- a set of symbols for **constants**, **predicates**, and **functions**
- **propositions** built from the above atoms
- **connectives** between simple propositions, eg: $\neg, \wedge, \vee$
- The **logical consequence** relation: $\Gamma \vdash \Delta$

The learning algorithm for logic-based AI is studied under the topic of ILP (inductive logic programming), which is a well-established field (cf my tutorial [11] or de Raedt's textbook [7]). ILP performs **combinatorial search** in the symbolic space of logic formulas, and is very

slow. The gradient descent of neural networks is much faster. In the following we examine the prospects of combining these 2 approaches.

Our idea is to transfer the structure of logic to neural networks. The logic structure can be broken down into:

- propositional-level structure
  (the mental state $\boldsymbol{x}$ is represented as a **set** of propositions $p_i \in \boldsymbol{x}$)
- sub-propositional structure

2.1 Propositional-level structure

Decomposition of the mental state into propositions

The mental state $\boldsymbol{x}$ may be broken down into a conjunction of propositions:

$$\boldsymbol{x}_1 = \text{ I'm attending a seminar } \wedge \text{ I feel hungry } \wedge .... \tag{11}$$

There could be another mental state:

$$\boldsymbol{x}_2 = \text{ I'm riding the subway } \wedge \text{ I feel hungry } \wedge .... \tag{12}$$

It would be highly inefficient if x_1 and x_2 are considered completely different states (without decomposition).

Boolean algebra

A **Boolean algebra** is a structure with:

$$\mathcal{B} = (\, A, \overbrace{\wedge, \vee}, \neg \,) \tag{13}$$

(underlying set: A; binary ops: $\wedge, \vee$; unary op: $\neg$)

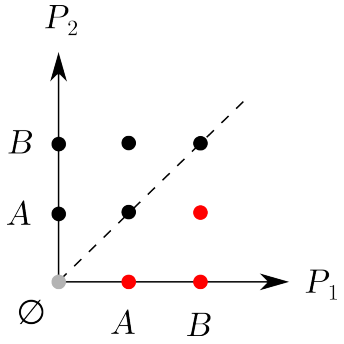
From my experience in logic-based AI, the most essential aspect of the Boolean structure (especially after introducing fuzzy-probabilistic reasoning) is this **commutativity** relation:

$$\forall a, b \in \mathcal{B}. \qquad a \wedge b = b \wedge a \tag{14}$$

Mapping propositions → vector space

Our gaol: map a set of propositions (*order is unimportant*) to a vector space.

Consider the simplest situation: P_1 and P_2 are two **containers** for propositions, the mental state is (P_1, P_2). Each proposition can be either A or B. We can arrange the placement of elements of $P_1 \times P_2$ in this way:



(15)

For example, the 3 red dots represent $\{A\}$, $\{B\}$, and $\{A, B\}$. The diagonal elements are not needed, because (A, A) etc. are *redundant* repetitions. In other words, all the propositional combinations are:

$$\{ \text{ lower-triangular elements } \} \setminus \text{ diagonal } \cup \varnothing \tag{16}$$

More abstractly, elements in this "symmetric space", eg. $(p_1, p_2, ..., p_n)$, are *invariant* under actions of the **symmetric group** S_n, n is finite.

In high dimensions, this space-saving scheme is extremely efficient: After symmetrization, a "corner" of the hypercube has a volume that is only $1/n!$ of the original cube. Even as the **curse of dimensionality** grows exponentially, $n!$ is well sufficient to offset it.

Now consider the **functions** that live on this symmetric space; How are their structures affected? Before symmetrization, the functions have domains:

$$\boldsymbol{F} : \mathbb{X} \to \mathbb{X} \quad = \quad \mathbb{P}^n \to \mathbb{P}^n \tag{17}$$

where $\mathbb{P}$ is the space of a single proposition. After symmetrization:

$$\boldsymbol{F} : \text{sym}(\mathbb{P}^n) \to \text{sym}(\mathbb{P}^n) \tag{18}$$

The symmetric space $\text{sym}(\mathbb{P}_1, \mathbb{P}_2, ...\mathbb{P}_n)$ possesses an **order**. That is to say, if the input to a function is $(p_1, p_2, ..., p_n)$, we need to **sort** the p_i's according to a certain order on $\mathbb{P}$, before calling $\boldsymbol{F}$ to calculate.

Sorting in $\mathbb{P}$ is simple, because each $p \in \mathbb{P}$ already has a **location** in vector space (that is learned via deep learning). $p_1 > p_2$ can be defined by a **cone** in the vector space, and then the order in $\mathbb{P}^n$ can be given by **lexicographic ordering**.

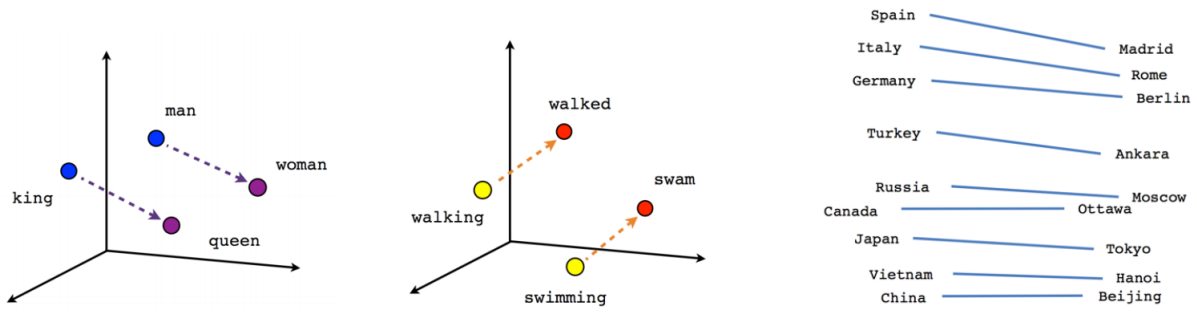
The algorithm in this section is very simple, yet it can achieve a $1/n!$ improvement. It is very worth implementing to see its actual performance ☺

2.2 Sub-propositional structure

This part is more complicated. Traditionally the structure of predicate logic is so esoteric from the point of view of mathematics that it hindered the progress of logic and logic-based AI.

Semantic distance and failure of compositionality

Word2Vec[6] embeds words into vector space with "semantic distance":



(19)

After Word2Vec, a natural next step is to represent **sentences** with semantic distance. One popular approach (eg [1]) is via **tensor products**. For example:

$$\textit{"I love you"} \quad \Rightarrow \quad \mathrm{i} \otimes \mathrm{love} \otimes \mathrm{you} \tag{20}$$

but this approach has serious problems: For example, pronouns such as "you" refer to various entities and must be resolved using rather complicated procedures, known as "abductive interpretation" in logic-based AI. Without such interpretations the word-entity correspondence becomes very inaccurate.

A logic based on "features"

Assuming that the compositionality problem can be bypassed, perhaps with logical atoms referring directly to entities (no pronouns, metaphors, etc), we may represent sentences by a "conjunction" of features (different from the $\wedge$ for propositions), eg:

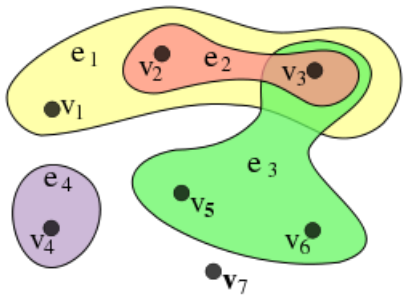
$$\begin{aligned} \boldsymbol{x} &= \textit{I feel hungry} \\ &= \textit{me} \cap \textit{physiology} \cap \textit{food} \cap \textit{negative-emotion} \cap .... \end{aligned} \tag{21}$$

This is like using a series of binary choices (= features) to determine an idea, like the game "20 Questions".

OpenCog's logic based on hypergraphs

To represent a complex scenario with many inter-relations, without using pronouns, perhaps **hypergraph** is a good choice. This is the idea behind OpenCog's representation.

A hypergraph is also called a "set system". Any hypergraph can be represented as a set of edges, each edge being an element of the power set of the nodes. Eg:



(22)

This representation is convenient for our purpose: Each edge can be represented as 1 proposition. The mental state $\boldsymbol{x}$ = a hypergraph = a set of propositions. We can use the symmetric space proposed above to represent the mental state.

Each proposition (edge) is a product of atomic concepts (nodes), we may further fix #(nodes) per edge to, say, 3 or 4:

$$\boxed{\text{proposition}} \quad \boldsymbol{p} = c_1\ c_2\ c_3\ c_4 \tag{23}$$

For example $c_1\ c_2\ c_3$ could be like subject-verb-object in natural language, where c_1 and c_3 are from the same underlying set of concepts; c_4 may express some auxiliary meaning, etc. We don't need to be very precise about this, as the actual representation can be figured out via machine learning. This is the age of *post-modern** AI!

"Linkage" phenomena in predicate logic

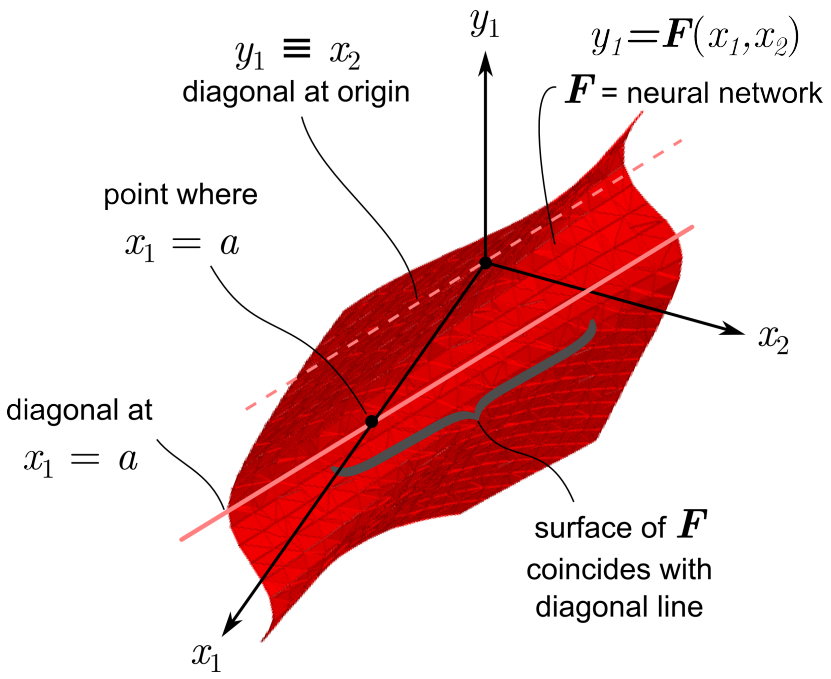
Next we consider **variable substitution**, a key feature of predicate logics. For example:

$$\forall X\ \forall Y\ \forall Z. \quad \text{grandfather}(X, Z) \leftarrow \text{father}(X, Y) \wedge \text{father}(Y, Z) \tag{24}$$

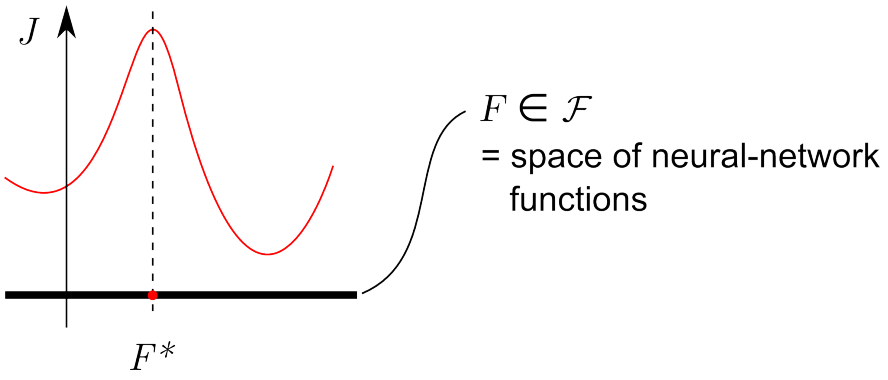
$$\boldsymbol{F}: \ (x_1, ..., x_i, ..., x_n) \mapsto (y_1, ..., y_j, ..., y_n) \tag{25}$$

$$\boldsymbol{F}: \begin{cases} y_j \equiv x_i, & \text{if } \boldsymbol{x} = \hat{\boldsymbol{a}} \text{ for some coordinates except } x_i \\ y_h \equiv x_k, & \text{if } \boldsymbol{x} = \hat{\boldsymbol{b}} \text{ for some coordinates except } x_k \\ \text{... etc ...} & \\ \text{is free otherwise} & \end{cases} \tag{26}$$

* With "modern" meaning "structuralist", as in designating structural elements such as "NP (noun phrase)" in phrase-structure grammar or the "is-a" link in old-fashioned AI.

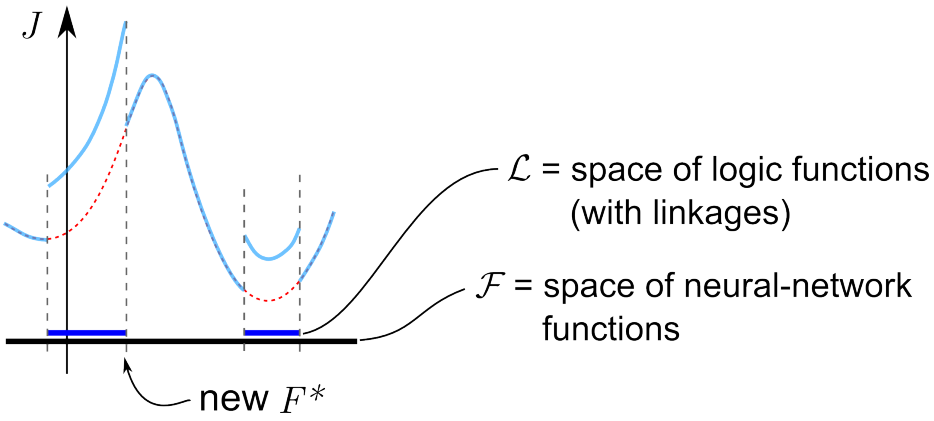


(27)



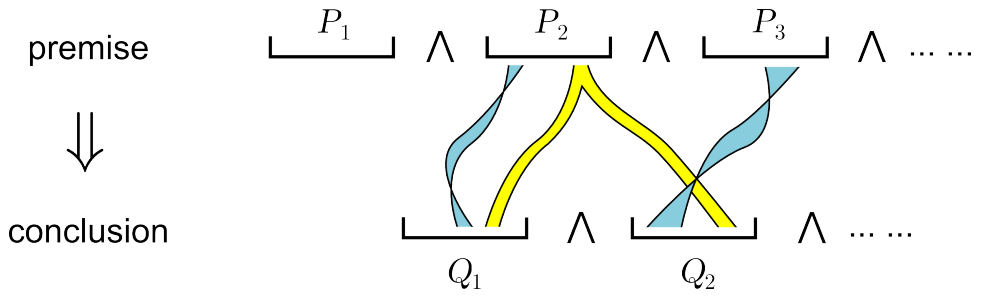
(28)

而我们需要的是新的 back-prop，它在「混合」的 function space 上 optimize（至於 exactly 怎样混合我还不清楚），J 在 logic 的 sub-space 上分数较高：



(29)

Learning algorithm with linkages



(30)

Data: training data?
Result: learn linkages
initialization;
while *not at end of this document* **do**
  read current;
  if *identity seems to exist* **then**
    learn the identity;
    while interleaving with dual-brain normalization;
  else
    find next identity;
  end
end

How about relation algebra?

Personally I think **relation algebra** [8] [5] is closer to human natural language, but the standard form used in mathematical logic research is first-order logic (FOL). This is however not of the essence, because all logics are basically easily interconvertable. In the following we concentrate on FOL.

$$\text{father} \circ \text{father} = \text{grandfather} \tag{31}$$

实际的推导是这样的：

$$\begin{aligned} \text{John father Pete} \\ \text{Pete father Paul} \\ \text{John father} \circ \text{father Paul} \end{aligned} \tag{32}$$

这时要 代入 上面的等式才能得出结论：

$$\text{John grandfather Paul} \tag{33}$$

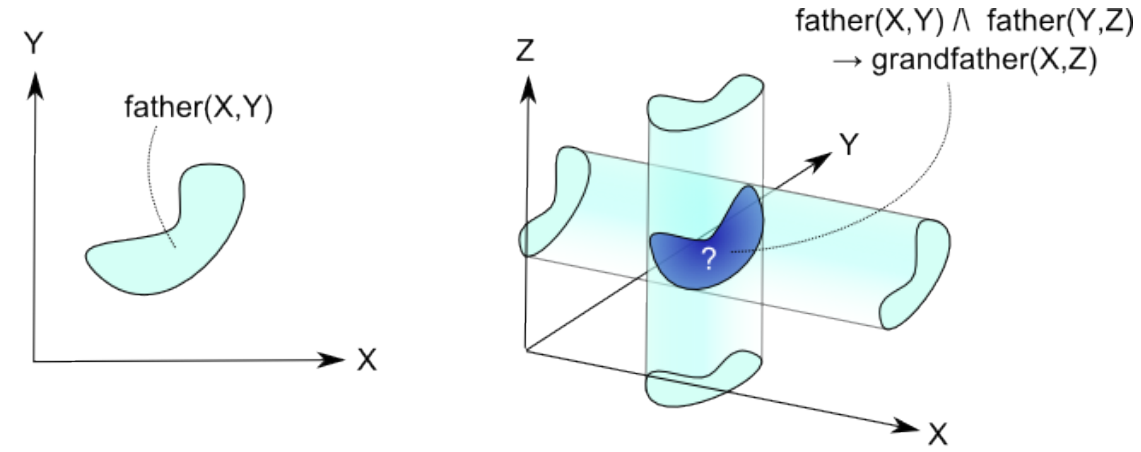
Fractal structure?

Sub-propositional structure – summary

$$\frac{\text{propositional calculus}}{\text{Boolean algebra}} = \frac{\text{predicate calculus}}{\text{cylindric algebra}} \tag{34}$$

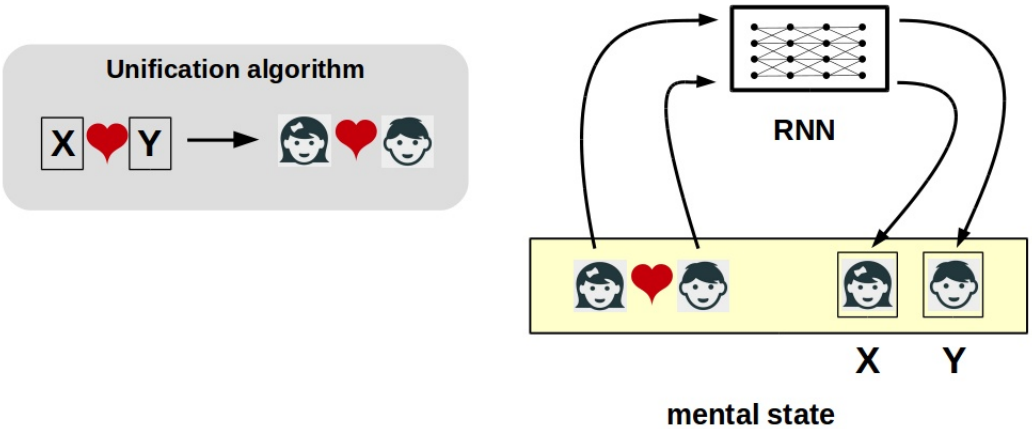
Higher-order logic (HOL) 的代数化可以用 **topoi theory** 给出 [3] [4]：

$$\frac{\text{intuitionistic propositional calculus}}{\text{Heyting algebra}} = \frac{\text{higher-order logic}}{\text{elementary topos}} \tag{35}$$



(36)

Trading space with time



(37)

Cartesian-closedness

$$\boxed{\text{product}}\ A \times B \quad \text{和} \quad B^A\ \boxed{\text{exponentiation}} \tag{38}$$

在逻辑中这是指：

$$A \wedge B \quad \text{和} \quad A \to B \tag{39}$$

3 Conclusion

Acknowledgement

Thanks to Ben Goertzel for discussions on the AGI mailing list. Ben first pointed out the advantage of neural network learning over inductive logic learning, which prompted me to research their relationships. Thanks also to Juan Carlos Pinto for personal discussions on the structure of neural networks.

References

1. Coecke, Sadrzadeh, and Clark. Mathematical foundations for distributed compostional model of meaning. *Linguistic Analysis 36, 345-384*, 2010.
2. Robert Gilmore and Marc Lefranc. *The topology of chaos: Alice in stretch and squeezeland.* Wiley-VCH, 2011.
3. Joachim Lambek. Categorical versus algebraic logic. In Andréka, Monk, and Németi, editors, *Algebraic logic*, pages 351–360. North-Holland, 1988.
4. Saunders MacLane and Ieke Moerdijk. *Sheaves in geometry and logic – a first introduction to topos theory.* Springer, 1992.
5. Roger Maddux. *Relation algebras.* Elsevier, 2006.
6. Mikolov, Sutskever, Chen, Corrado, and Dean. Efficient estimation of word representations in vector space. *Proceedings of workshop at ICLR*, 2013.
7. Luc De Raedt. *Logical and relational learning.* Springer, 2008.
8. Gunther Schmidt. *Relational mathematics.* Cambridge, 2010.
9. Stephen Smale. Differentiable dynamical systems. *Bulletin of the American Mathematical Society*, 1967.
10. Tamás Tél and Márton Gruiz. *Chaotic dynamics: an Introduction based on classical mechanics.* Cambridge, 2006.
11. King Yin Yan. ILP tutorial `https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/genifer/Genifer-induction(30July2012).pdf`.
12. King Yin Yan, Juan Carlos Kuri Pinto, and Ben Goertzel. Wandering in the labyrinth of thinking – a cognitive architecture combining reinforcement learning and deep learning. (to be submitted AGI-2017).